Migrating an Existing Service

This guide shows how to prepare a web service to run in a container on Cloud Run. It does not cover data migration.

Listen on the port defined by the `PORT` environment variable

Your service must listen on the port specified by the `PORT` environment variable, as specified in the container runtime contract (https://cloud.google.com/run/docs/reference/container-contract).

You can add default values in case a `PORT` environment variable is not defined. This simplifies development because it allows you to omit this variable in `docker run` commands.

Listen on the `PORT` variable by modifying your code

Modify your service to support prioritized port configuration from the `PORT` environment variable. This approach reduces operational overhead.

1. When a service starts up, all environment configuration is made available. Check to see if the `PORT` environment variable has been set. This is always present on Cloud Run, but might not be available if running locally.
2. If the `PORT` environment variable is set, your service must listen on that value.
3. If the `PORT` environment variable is not set, you can handle this as an error or provide a fallback port value.

For sample code that performs these steps, see the build and deploy (https://cloud.google.com/run/docs/quickstarts/build-and-deploy) quickstart.

For PHP and other languages that do not have a direct HTTP interface, the port configuration is handled via a web server (such as Apache). If the web server cannot be configured to listen to an environment variable, you may need to use a customized container entrypoint, which is described next.

Setting existing port configuration via the container entrypoint

If the service already has a mechanism to configure the port, such as another environment variable or a configuration file, you can avoid refactoring the application code. Instead, add startup logic that maps the `PORT` environment variable to your configuration.

1. Open the service Dockerfile. If one does not exist, add a `Dockerfile` (#add-dockerfile) first, then continue these steps.
2. Create a container entrypoint (https://docs.docker.com/engine/reference/builder/#entrypoint) shell script to map the `PORT` environment variable to your configuration setting. For example, create a new file named `dockerfile-entrypoint.sh` that has the following contents:

   ```
   #!/usr/bin/env bash
   export YOURAPP_PORT=":${PORT}"
   # Execute the rest of your ENTRYPOINT and CMD as expected.
   exec "$@"
   ```

   In the example above, a BASH script maps `PORT` to an environment variable named `YOURAPP_PORT`, which needs to be preceded by a colon (:), a common alternative port format.

   The final line, `exec "$@"` executes further entrypoint steps as configured in your `Dockerfile`, followed by the configured command (`CMD`) that starts your service.
3. Make sure your script is executable:

   ```
   chmod +x docker-entrypoint.sh
   ```
4. Copy your script into the container by adding this line to your `Dockerfile`:

   ```
   COPY docker-entrypoint.sh /docker-entrypoint.sh
   ```
5. Execute your script as part of the container entrypoint by adding this line to your `Dockerfile`:

   ```
   ENTRYPOINT [ "/docker-entrypoint.sh" ]
   ```

   If your Dockerfile already defines an entrypoint (or extends from a base image that defines one) you must preserve the existing entrypoint

```
ENTRYPOINT [ "/docker-entrypoint.sh", "/existing-entrypoint.sh" ]
```

Remove reliance on local filesystem for persistent storage

Check your application code for reliance on local filesystems, and replace it with file storage on Cloud Storage (https://cloud.google.com/storage) or data storage such as Cloud Firestore (https://cloud.google.com/firestore) or Cloud SQL (https://cloud.google.com/sql).

Add a `Dockerfile`

In order to wrap your service in a container, you should use a Dockerfile to define the operating environment. The build and deploy quickstart (https://cloud.google.com/run/docs/quickstarts/build-and-deploy#containerizing) shows some basic Dockerfiles you can use to get started.

Read more about customizing your `Dockerfile` on the Developing your service (https://cloud.google.com/run/docs/developing#add-dockerfile) page.

Tune for concurrency and memory limits

Learn more about how concurrency works (https://cloud.google.com/run/docs/about-concurrency) then optimize your service concurrency support in development tips for tuning concurrency (https://cloud.google.com/run/docs/tips#tuning-concurrency)

Make sure your memory limit (https://cloud.google.com/run/docs/configuring/memory-limits) is high enough for your application to run, including capacity for any temporary file storage your service requires.

Send logs to stdout, stderr, or /var/log

Cloud Run automatically uses Stackdriver to aggregate and review your logs, as described in the logging page (https://cloud.google.com/run/docs/logging).

Except as otherwise noted, the content of this page is licensed under the Creative Commons Attribution 4.0 License (https://creativecommons.org/licenses/by/4.0/), and code samples are licensed under the Apache 2.0 License (https://www.apache.org/licenses/LICENSE-2.0). For details, see our Site Policies (https://developers.google.com/terms/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated November 14, 2019.